R Programming Challenge

Haigang Liu

July 31, 2016

Introduction

In this article, a few R programming challenges are presented, most of which are based on our lectures. I put much emphasis on loops, functions, plots and simulation, since they are the building blocks of advanced topics.

Solutions are given at the end of each question. Note 1) there is not only one way to solve a given problem, and it is particularly true when it comes to programming 2) do not skip my comments.

If any of you use Python, I recommend to try out `numpy` and `pandas`. They are optimized for matrix computation.

Question 1

In a pollution control example in class, we talked about finding the optimal sample size for a given threshold. In this case, the population distribution of $Y \sim N(48, 100)$, and thus $\bar{Y} \sim N(48, 100/n)$. We calculated in class that $P(\bar{Y} > 50) = 0.0228$. We also know the fact that $P(\bar{Y} > 50)$ is decreasing as n increases. Hence, how large should the sample size n be so that $P(\bar{Y} > 50) < 0.01$?

Solution The path is clear since one can find $P(\bar{Y} > 50)$ for all n. Since we know it is decreasing as n increases, it suffices for us to find out the threshold. Beyond that threshold, $P(\bar{Y} > 50)$ will be always smaller than 0.01. The R code is given as follows. The comments start with pound sign.

```
# you can assign mean and variance values by equal sign
# split your arguments with semicolon
mu = 48; sigma = 10

# find_probability_for_given_n is a function that takes one argument--sample size
# pnorm is a function to find out probabilities under normal distribution
find_probability_for_given_n = function(n){
  prob = 1 - pnorm(50, mean = mu, sd = sigma/sqrt(n))
  return (prob)
}

# You search from n = 1, and calculate find_probability_for_given_n for each n
# stop when finding the n that satisfies P (Y_bar > 50) < 0.01
```

```
# The desired sample size is 136.

for (i in 1:1000){
  if(find_probability_for_given_n(i) <0.01){
    print(i)
    break
  }
}
```

Question 2

To learn the behavior of a random variable, it would be a neat idea to plot its pdf, provided that the pdf exists and readily available. How to plot a given pdf in R?

Solution In this case, we use normal v.s. Cauchy as an example (Figure 1), both of which have known pdf. Also, we want to compare them by plotting them in one graph.The R code to do this is given after the graph.

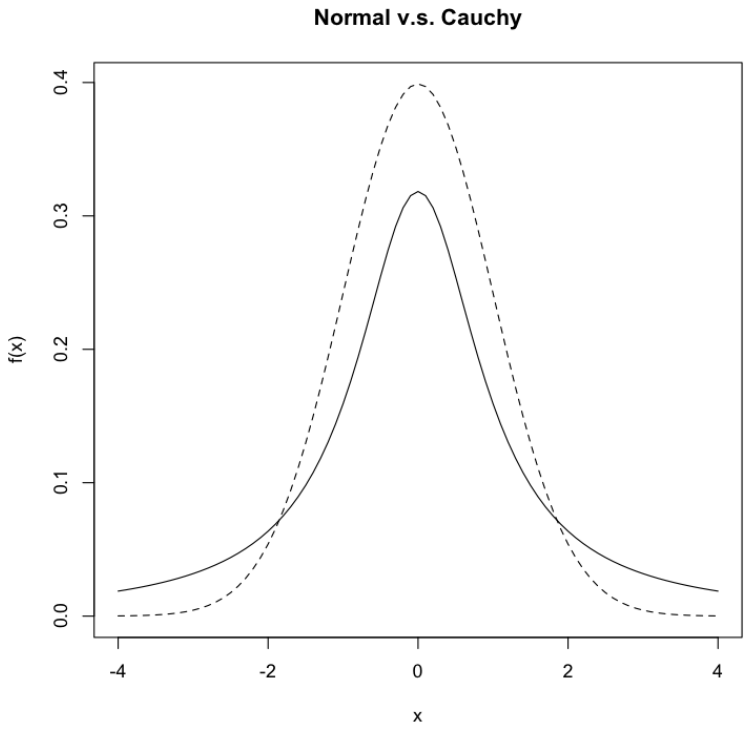

Figure 1: Cauchy v.s. Normal

```
# c(1,1) is for one graph, you can use c(1,2) to display two pics side by side.
par(mfrow = c(1,1))

# We calculate each function values for x from -4 to 4
# and by connecting these points computer renders a smooth curve
# x, cauchy and normal are all vectors, or arrays.
x = seq(from = -4, to = 4, by = 0.1)
cauchy = 1/(pi*(1+x^2))
normal = 1/(sqrt(2*pi))*exp(-0.5*x^2)
range_of_fx = range(0, cauchy, normal)

# lines() is a function to help us to put one graph on top of another
# if you use plot in both cases, you'll get two graphs seperately
# in other words, each time you use ``plot'', you initialize a new canvass.
plot(x, cauchy, type = "l", ylab = "f(x)", ylim = range_of_fx,
main = "Normal v.s. Cauchy")
lines(x, normal, lty = 2)
```

Question 3

In class, we came across a pivotal quantity whose pdf is defined as $f_U(u) = nu^{n-1}$, for $u \in (0, 1)$. We can find three types of confidence interval, i.e., $(0, a)$, $(b, 1)$ and (a, b). And I claimed that $(b, 1)$ type is the shortest, therefore, the best, in this comparison. However, I failed to show this analytically. Can we solve this numerically?

Solution The length, or width of confidence interval is a function of α. It suffices for us to plot all three functions in one graph. If there exists one type that is uniformly smaller than others for all α, then that type is considered the best. As Figure 2 suggests, $(b, 1)$, i.e., L_3 is superior than two other candidates. The R code is given after the graph.

```
# In programming, you use all upper case letter to define a constant
# google ``style guide in r'', and you can find how to name variables and functions
# you can actually any sample size. The conclusion stays the same.
SAMPLE_SIZE = 2

#three width functions based on three types of confidence intervals
L_1 = function(a, n = SAMPLE_SIZE){
  l1 = (1-a/2)^(1/n) - (a/2)^(1/n)
  return(l1)
}

L_2 = function(a, n = SAMPLE_SIZE){
  l2 = (1-a)^(1/n)
  return(l2)
}

L_3 = function(a, n = SAMPLE_SIZE){
  l3 = 1 - (a)^(1/n)
  return(l3)
```

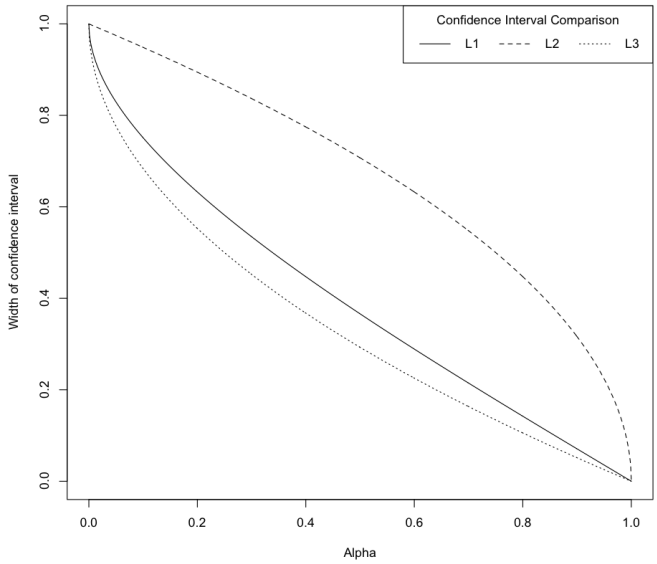

Figure 2: Width function for three types of confidence intervals

```
}

# calculate L1(a), L2(a) and L3(a) for every alpha in (0,1)
# and plot them out to compare,
# it is suggested L3 is uniformly better, which is consistent with our speculation.
alpha_support = seq(from = 0, to = 1, by = 0.001)
plot(alpha_support, L_1(alpha_support), type= "l", lty =  1,
ylab = "Width of confidence interval", xlab = "Alpha")
lines(alpha_support, L_2(alpha_support), lty = 2, ylab = "L2")
lines(alpha_support, L_3(alpha_support), lty = 3, ylab = "L3")
legend("topright",  title="Confidence Interval Comparison",
       c("L1","L2","L3") , lty=1:3, horiz=TRUE)
```

Question 4

One nice feature of R is built-in simulation functions. The idea of simulation is to get a random sample which follows a prescribed distribution. For example, in R you can get a sample of 100 standard normal random variables by `rnorm(100)`.

We can use this feature to do Monte Carlo simulation, as suggested in Question 5. An even simpler application of simulation is this: sometimes variable transformation is hard to prove analytically, but they are easy to program. For example, suppose you do not know what is the distribution of $-\ln Y$, in which $Y \sim U(0,1)$. You can first generate 1000 random uniform variables, take their log, and plot in a histogram, from which you might get some inspiration

what $-\ln Y$ is.

We now know that $-\ln Y$ follows an exponential distribution from variable transformation. As you can see in Figure 3, the simulated $-\ln Y$, which is displayed in histogram, are close to the pdf of exponential distribution. The code of simulation and histogram is given as follows.

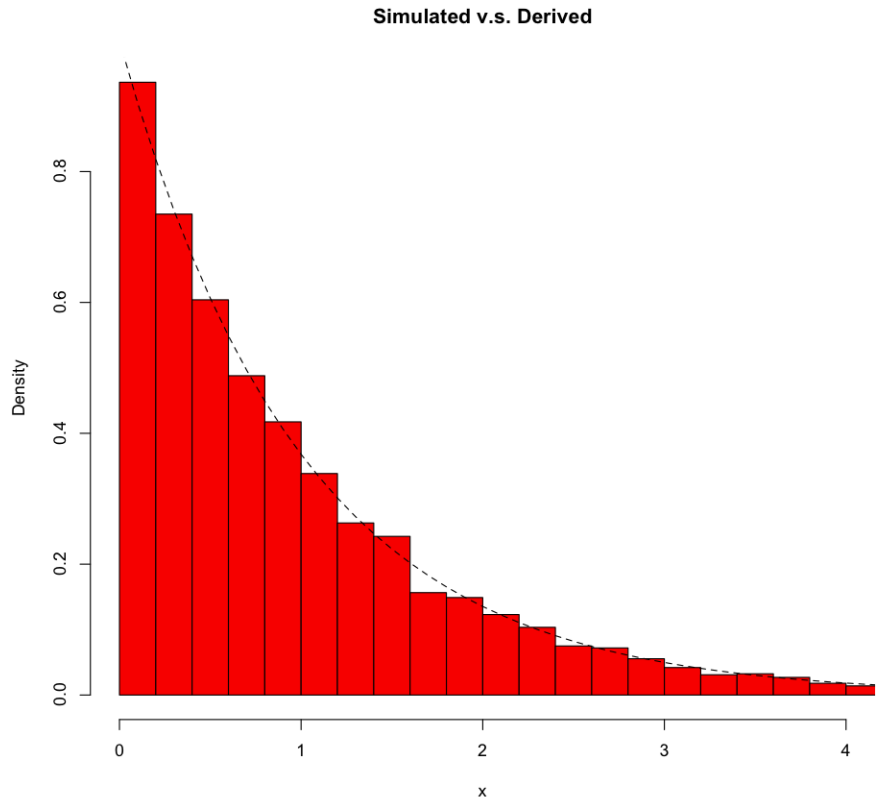

Figure 3: Derived v.s. Simulated

```
par(mfrow = c(1,1))

# runif is the command to get random uniform variables
# you can change its lower and upper limits by tweaking min and max
sample_size = 10000
random_variable_uniform = runif(n =sample_size, min = 0, max = 1)
transformed_varible = -log(random_variable_uniform)

derived_curve_x = seq( from = 0, to =10, by = 0.1)
derived_curve_y = exp(-derived_curve_x)

# in hist function, nclass controls how many bins are there.
# if freq = TRUE, then y axis will be frequencies, rather than density.
```

```
hist(transformed_varible,xlab = "x", nclass= 50,
main = "Simulated v.s. Derived", col = "red", freq = FALSE, xlim = c(0,4))
lines(derived_curve_x,derived_curve_y, lty = 2)
```

Question 5

We know that $\hat{p}$, the sample proportion, follows a normal distribution as n goes to infinity, or at least large enough. Hence, to use CLT, what sample size is proper? The answer is suggested in Figure 4, in which the sampling distribution of $\hat{p}$ is pretty close to normal when $n = 40$. It is not quite safe when $n = 10$ due to tail behavior.

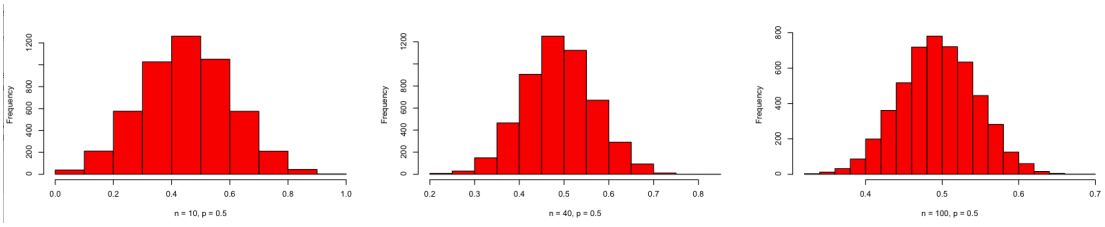

Figure 4: The sampling distribution of $\hat{p}$ for different n when population proportion is 0.5. From left to right: 10, 40, 100.

However, we just considered the case in which population proportion is assumed to be 0.5, i.e., something like a fair coin. One must also notice that $\hat{p}$ follows a skewed distribution when p, the population parameter, is not 0.5. The sample size should be larger for those with skewed population distribution to converge to normal. And this is suggested in the following graph, in which $p = 0.1$.

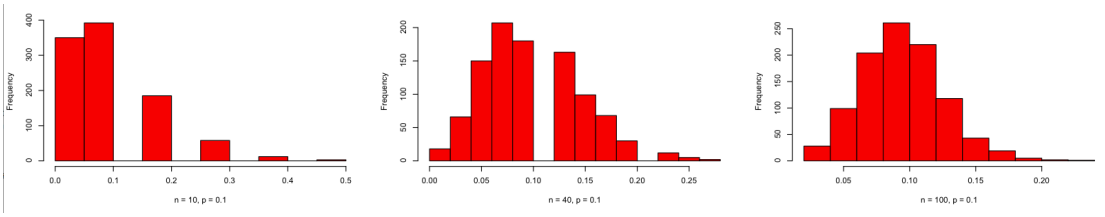

Figure 5: The sampling distribution of $\hat{p}$ for different n when population proportion is 0.1. From left to right: 10, 40, 100.

In Figure 5, it is clear that the sampling distribution is far from normal distribution when $n = 10$ or $n = 40$. It does not start converging until $n = 100$. Bottom line is, the sample size $\hat{p}$ requires to converge, is dependent on p, the population proportion. The closer to symmetric, the faster the converging is.

We established all the above based on simulation. The scheme to implement the simulation is described as follows. Note that in fact 6 scenarios are discussed. We use $n = 10$ and $p = 0.1$ for demonstration purpose.

- Sample 10 observations from Bernoulli(0.1). Denote this as sample K.

- Gather $\hat{p}$ from sample K.
- Repeat step 2 for m times. We often use $m = 1000$.
- Plot 1000 $\hat{p}$ in a histogram.
- Repeat all the above for $n = 40$ and $n = 100$
- Repeat all the above for $p = 0.5$.

The R code to implement the steps above is given as follows.

```
# m is how many trials we have in Monte Carlo Simulation
m = 1000

# generate_p_hat is a function to get a p-hat based on designated sample size
# for example, it can sample 10 Bernoulli variables and get a sample proportion
generate_p_hat = function(sample_size, success_rate){
  bernoulli_variables = NA
  i = 1
  while (i < sample_size + 1){
    bernoulli_variables[i] = rbinom(size = 1, n = 1, p = success_rate)
    i = i + 1
  }
  p_hat = sum(bernoulli_variables)/sample_size
  return(p_hat)
}

# generate_m_p_hats is a function based on last one
# which repeats last function for 1000 times, which is m in this case.
generate_m_p_hats = function(sample_size, p){
  p_hats_vector = NA
  for (i in 1:m) {
    p_hats_vector[i] = generate_p_hat(sample_size, p)
  }
  return(p_hats_vector)
}

# plot 6 scenarios based on different sample size and different success rate.
# if p = 0.1, underlying distribution is asymmetric
# if p = 0.5, underlying distribution is symmetric, thus n = 40 we can use CLT already.

par(mfrow = c(2,3))
hist(generate_m_p_hats(10, 0.1), col = "red", xlab = "n = 10, p = 0.1", main = "")
hist(generate_m_p_hats(40,0.1), col = "red",  xlab = "n = 40, p = 0.1", main = "")
hist(generate_m_p_hats(100,0.1), col = "red",  xlab = "n = 100, p = 0.1", main = "")

hist(generate_m_p_hats(10, 0.5), col = "red", xlab = "n = 10, p = 0.5", main = "")
hist(generate_m_p_hats(40,0.5), col = "red",  xlab = "n = 40, p = 0.5", main = "")
hist(generate_m_p_hats(100,0.5), col = "red",  xlab = "n = 100, p = 0.5", main = "")
```